Weaving Variability into Domain Metamodels

Brice Morin, Gilles Perrouin, Philippe Lahire, Olivier Barais, Gilles Vanwormhoudt, Jean-Marc Jézéquel

Weaving Variability into Domain Metamodels*

Brice Morin[1], Gilles Perrouin[2], Philippe Lahire[3], Olivier Barais[1,4], Gilles Vanwormhoudt[5], and Jean-Marc Jézéquel[1,4]

[1] INRIA, Centre Rennes - Bretagne Atlantique, Equipe Triskell, F-35042 Rennes Cedex
[2] University of Luxembourg, LASSY, L-1359 Luxembourg-Kirchberg, Luxembourg
[3] I3S Nice-Sophia Antipolis, Equipe Rainbow, F-06903 Sophia-Antipolis Cedex
[4] IRISA, Université de Rennes1, Equipe Triskell, F-35042 Rennes Cedex
[5] Institut Telecom / LIFL, Université de Lille 1, F-59655 Villeneuve d'Ascq Cedex

Abstract. A Domain-Specific Modeling Language (DSMLs) describes the concepts and their relationships of a particular domain, in a metamodel. Using a DSML, it is possible to describe a wide range of different models. These models often share a common base and vary on some parts. Current approaches tend to distinguish the variability language from the DSMLs themselves, implying greater learning curve for DSMLs stakeholders and a significant overhead in product line engineering. We propose to consider variability as an independent aspect to be woven into the DSML to introduce variability capabilities. In particular we detail how variability is woven and how to perform product line derivation. We validate our approach through the weaving of variability into two different metamodels: Ecore and SmartAdapter, our aspect model weaver. These results emphasize how new abilities of the language can be provided by this means.

1 Introduction

In an always more competitive environment, the ability for a company to rapidly propose new products or variations of existing products is the key to meet user requirements. However, proposing a wide range of different products is risky: products should be designed, validated, implemented rapidly, at a low cost. The Software Product Line [1] (SPL) community proposes techniques and tools in order to engineer families of related products. The main idea behind SPL is to capture the commonalities of the different products as well as the specificities (variability) of each particular product. In this paper, we focus on Model-Driven SPL [2] where the product line itself and the derived products are models. The models are derived using Model-Driven Engineering techniques, such as model transformation, model composition or aspect model weaving.

Several approaches exist to describe SPLs : *i)* to use a general-purpose metamodel like the UML [3] including the concepts allowing designers to describe the variability of a model, *ii)* to build a metamodel without variability and then extend it in order to include the necessary variability. The first category allows the domain stakeholder to directly design a family of products thanks to the expressiveness provided by the metamodel. The second category allows designers to focus on the domain itself but not on its variability and then to update it in order to include

* This work was realized in the context of the MOVIDA and the FAROS projects, funded by the ANR (French National Research Agency)

the needed variability. The main drawback of this approach is that the variability should be included manually. For example, this is what we have done when we included variability into our aspect model weaver "SmartAdapters" [4], in order to be able to design more flexible and more reusable aspect models.

In this paper, we propose a reusable variability aspect, defined at the metamodel level, describing the variability concepts and their relationships independently from any domain metamodel. Using Aspect-Oriented Modeling [5] (AOM) techniques, this aspect can be woven into a given domain metamodel to include variability. More precisely, we use our aspect model weaver, SmartAdapters, to weave this variability aspect. This makes possible the integration of variability in a semi-automatic way into a wide range of domain metamodels.

We demonstrate our approach by introducing the variability *i)* into EMF, which is the *de-facto* standard integrated within Eclipse to define metamodels (similar to class diagrams) and *ii)* into SmartAdapters itself. However, our approach is generic and can be applied to any metamodel conforming to Ecore/EMOF, which includes the UML, to extend it with variability mechanisms. This demonstrates that decoupling the description of variability from the domain allows addressing, with a minimal additional effort, the two categories we mentioned.

This paper is organized as follows. In Section 2 we present a generic variability metamodel. In Section 3, we illustrate how we can introduce variability by hand into an excerpt of the EMF metamodel. In Section 4 we present our model weaver SmartAdapters and give an overview of the variability aspect associated to the variability metamodel presented in Section 2. Then we propose in Section 5 to apply the variability aspect to the SmartAdapters metamodel. Section 6 details how we can take advantage of SPL techniques in order to derive models with respect to variability woven at the metamodel level. Section 7 outlines some relevant research in the field and Section 8 wraps up with conclusions as well as discusses some interesting future perspectives.

2 Variability

In this section, we present a generic variability metamodel inspired from [2]. It relies upon formal studies of variability management in SPL [6, 7] and in particular feature diagramming [8–11], which is a very popular notation in this community. Despite their wide acceptation, many variants have been proposed and there is no *de-facto* standard for feature diagrams. One interesting result of these formal studies is that it is possible to extract a pivot abstract syntax subsuming the expressiveness of these existing notations hence forming a universal basis independent of any peculiar notation. This abstract syntax is the source of our metamodel (see [2] for details).

Our generic variability metamodel (See Figure 1) is a customization of the feature metamodel [2] tailored to describe variability amongst Ecore concepts. In particular the notion of feature hierarchy, which is very specific to feature diagram, has been omitted because this hierarchy is imposed by the target metamodel (see Section 6). The main metaclass is $PointOfVariability$ which provides the possibility to metamodel elements on which this class is woven to have variants. $PointOfVariability$ has a concrete sub-class called $VariabilityOfElement$ allowing a given domain concept to hold variability. Variability is described in

terms of boolean operators that describe the kind of variability relationships applies to elements. *And* operator holds true iff all the elements to which it applies are chosen (mandatory elements). Xor denotes an alternative (only one element have to be chosen) and *Or* at least one. *Opt* denotes the optionality of presence. Finally $Vp(i, j)$ [7] will return true iff at least i and at most j elements are chosen. This operator can embed the semantics of all other operators [6] and could hence be the unique operator provided. However, "classic" operators are more practical and well-known; They are therefore left for usability matters.
In addition in this metamodel we make the distinction between *homogeneous* and *heterogenous* operators. Homogeneous operators are associated to $VariabilityOf-Element$ and apply only on element of the same type (EClass, etc.). Heterogeneous operators are associated to $PointOfVariability$ and apply to elements of different types. The "choice" semantics is the same for homogenous and heterogeneous operators. However, we distinguish the hierarchy of operators associated to *homogeneousOperator* and *heterogenousOperator* because we support the idea that the domain expert should be able to add to its metamodel, only the expressiveness he needs. Thus he should be able to choose the suitable homogeneous as well as heterogeneous operators.

Additionally, it is possible to define constraints between points of variability, whether there are targeted by the same operator or not. These constraints are of two types: *requires* which implies that the required element have also to be selected if the requiring element is selected, and *mutex* which excludes that both referred elements are present in the same selection. There exists other kinds of constraints [2, 6, 7] but as there mostly informal (expressed in natural language) we decided to let the designer include them manually after weaving.

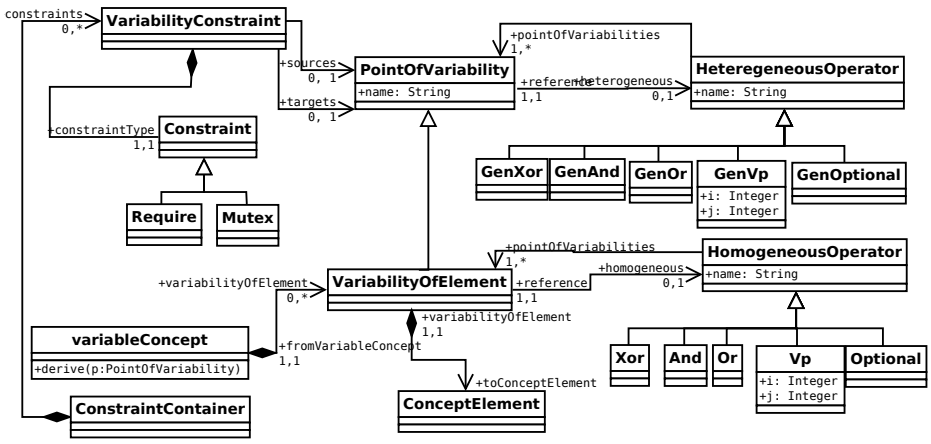

Fig. 1. The variability metamodel

3 Patterns for introducing variability

As noted by Haugen *et al* [12], there are two categories of techniques to introduce variability into languages (represented as metamodels); *amalgamated* and *separated*. The first one proposes to augment the metamodel with variability constructs while the second one keeps them distinct and relates them via simple referencing. We chose the first kind of approach because we want to clearly express variability among elements [12] and enable conformance checking in a standard way.

For example in EMF, if it is required that *i)* some classes, operations, or attributes are optional, and *ii)* some model elements are part of the same variant, and *iii)* alternative and constraints may exist among variants, then this information cannot be attached into the EMF metamodel and have to be put in a feature diagram or in a DSL as proposed by Haugen et al.

In this paper, our aim is to describe models containing variability, based on the concepts defined in the domain metamodel. In order to do that, we construct on demand a new metamodel MM' that integrates concepts from product-lines described in Section 2 and the domain-specific concepts of the MM. The main idea is to introduce a new meta-class on each association to capture the fact that a model element can be a variant as illustrated in Figure 2 for an excerpt of the EMF metamodel. As an example, it allows to capture the fact that a method $m1$ in a class A is optional and two methods $m2$ and $m3$ are alternatives as described in Figure 3.

Implications of this approach are that the variability aspect can also be standardized, and that DSMLs (MM) can be designed without variability mechanisms. These mechanisms are woven on demand to create MM'.

We still want to keep the possibility to specify that a class A has a method $m4$ which is not a variable part of the model. Consequently, we do not remove existing associations, but extend the domain metamodel with new class-associations

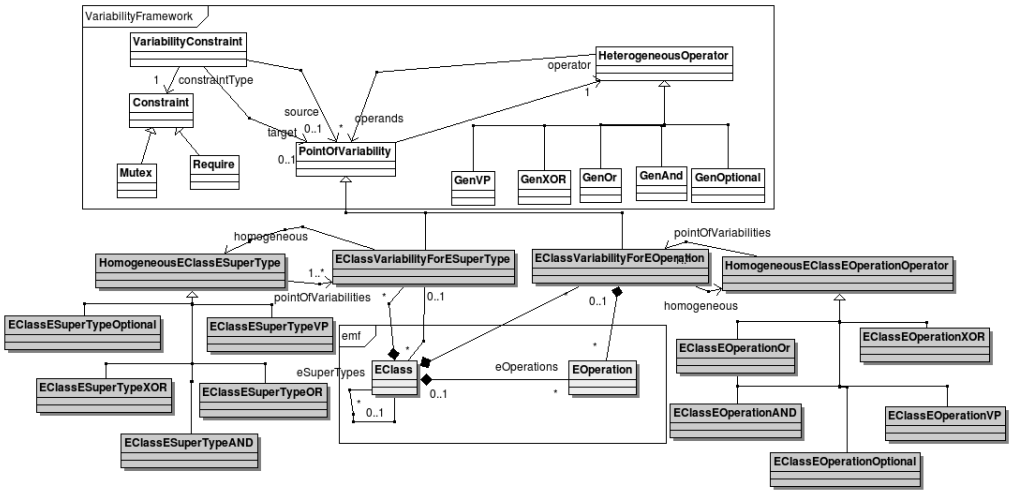

Fig. 2. EMF meta-model with Variability

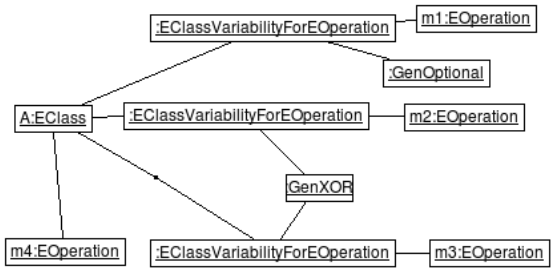

Fig. 3. EMF model with Variability

($EClassVariabilityForEOperation$ and $EClassVariabilityForESuperType$ in Figure 2), which capture the variability.

The idea of the pattern to introduce variability is to match each association and create new meta-classes for creating the connection between MM and the variability aspect. This way, the former metamodel MM is simply extended. Since all the existing meta-classes and their properties (from MM) are kept in MM', all the pre-existing models conforming to MM can easily be converted into models conforming to MM'.

4 Using SmartAdapter to Weave Variability

In the previous section, we presented a metamodeling pattern that provides a generic solution for extending a metamodel with variability. To ease the inclusion of variability into a wide range of metamodels or several parts of one metamodel, we propose to adopt an Aspect-Oriented Modeling approach. The main idea is to describe a variability aspect based on the previous pattern and weave this aspect into any metamodel. The use of an AOM approach provides several benefits: first, it enables decoupling the description of variability from any particular metamodel making it reusable; it also enables integration of variability in a semi-automatic way; lastly, it enables keeping the design of metamodel and variability separate, making their evolution easier to manage.In the following we briefly describe our SmartAdapters AOM approach [4] including a presentation of its metamodel, where we will weave variability (Section 5). Then we present its use to describe the variability aspect and apply this aspect for introducing variability into Ecore.

SmartAdapters is a generic AOM approach. It relies on four key concepts : aspect model, graft model, target model and adaptations. An aspect model consists of *i)* a graft model that encapsulates a given concern, and *ii)* an abstract adapter that describes *where* (target model) and *how* (adaptations) the aspect model will be woven into other base models.

The metamodel describing the concepts of SmartAdapters is shown at Figure 4. This metamodel is not tied to a specific domain metamodel and can be customized to weave aspects into different kinds of model (provided that aspect and base model rely on the same domain metamodel). Here, since we are interested to weave aspects into any metamodel, we assume the customization of SmartAdapter to MOF/Ecore metamodels.

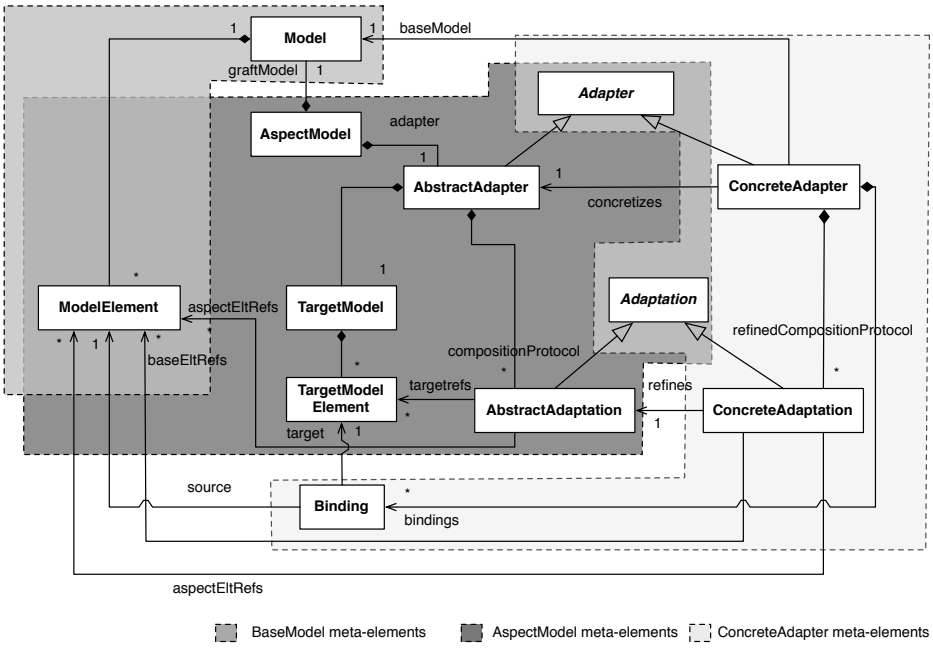

Fig. 4. The SmartAdapters metamodel

The target model (*TargetModel*) is an abstract interface between the aspect model (*AspectModel*) and any base models (*Model*). It is a model fragment that identifies the hooks required on the base model. It contains roles (*TargetModelElement*) that may be substituted, at binding time, by base model elements and structural constraints that every binding (a set of elements substituting the roles) should respect.

An abstract adapter (*AbstractAdapter*) is the composition protocol of an aspect model: it guides and controls the composition of the aspect, independently from any base model. It contains Adaptations (*AbstractAdaptation*) which are composition operations describing how to weave the aspect model into the target model. In a composition protocol, the designer can refer to any role from the target model or model element from the graft model, within the adaptations of the protocol.

The set of adaptations provides support for integrating graft models into any base model, by: *i)* **introducing** model elements *e.g.* a class into a package, *ii)* **modifying** properties (attributes and references) of a model element *e.g.*, a method signature, and *iii)* **merging** model elements *e.g.* two classes into a single one.

To actually weave an aspect model, an architect must design a concrete adapter (*ConcreteAdapter*). It specifies bindings (join points) between the target model and a given base model. Each binding (*Binding*) associates a target model element (*TargetModelElement*) to a matching base model element. Bindings could be specified by hand or automatically identified by a join point detection engine [13]. All the bindings contextualize the adaptations defined in the abstract adapter with concrete elements. Additionally, during the binding stage, the architect can

specify some other concrete adaptations (*ConcreteAdaptation*) to consider some properties specific to the base model.

Figure 5 shows the variability aspect. Basically, the graft model of the aspect contains the concepts from the feature metamodel presented in Section 2 and those from the metamodeling pattern presented in Section 3.

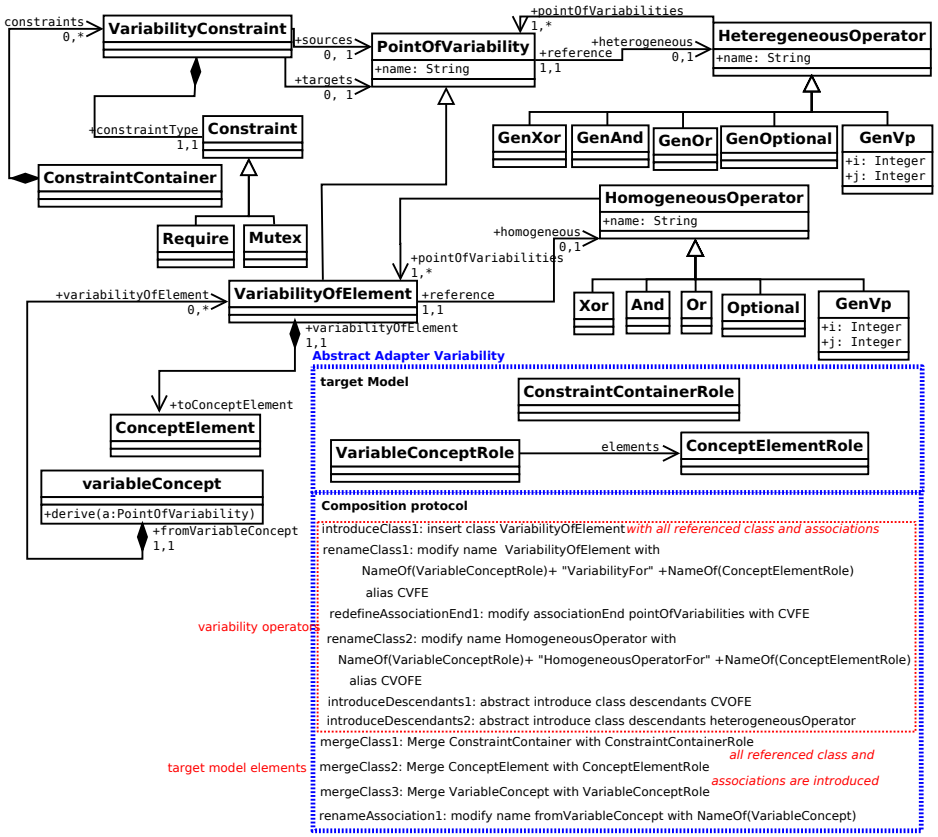

Fig. 5. The Variability Aspect including Composition Protocol

The target model contains roles to specify that three classes and one relationship must be present in a base model to apply the aspect. *ConstraintContainerRole* identifies the class in a metamodel where constraints for controlling variability must be attached. *VariableConceptRole*, *ConceptElementrole* and *elements* relationship identify a couple of linked classes in a metamodel where variability must be introduced.

The basic principle of the composition protocol is to: *i*) keep unchanged the relationship between the two classes (*VariableConceptRole* and *ConceptElementrole*) of the target model (to allow defining mandatory element), and *ii*) create a new relationship between these two classes, controlled by a variability manager, in order to allow defining variable elements.

More precisely the composition protocol mainly contains adaptations for introducing model elements (*insert*) which allow to introduce a given element (e.g. class *VariabilityOfElement*) with implicitly all elements that it references. When an element already exists in the base model (for example when the aspect is applied two times on the same model), it is not added a second time.

Another important remarks deals with the use of renaming (*modify name*). One relationship (*fromVariableConcept*) and two classes (*variabilityOfElement* and *VariabilityOperator*) are renamed. Main advantages in present situation is that n applications of the aspect in the same base model will create n samples of the same relationship or metaclass.

Finally for each application of the aspect we may choose the expressiveness of the variability. We simply select the descendant classes of *VariabilityOperator* after its renaming (the renamed class is accessible with the *alias* CVOFE). The choice is made accordingly to the base model (at composition time), so that the adaptation is abstract and will be defined in a concrete adapter (*ConcreteAdapter*).We may choose the same approach for the different types of constraints. This would be particularly interesting if we propose a larger set of constraint types. In order to reduce the complexity of the schema we decided to provide all the types of constraints (*Require* and *Mutex*) for each application of the aspect. In order to provide a centralized access to all constraints in a given element of the base model, we declare this element in the target model.

In our composition protocol we also propose to merge each of the three elements mentioned in the target model with one element of the graft model. This way, base model elements bound to target model elements now include their respective functionalities (*e.g.* the class(es) bound to *VariableConceptRole* will include the *derive* method and the association-end *variabilityOfElement*).

Figure 6 shows the concrete adapter to apply the variability aspect to EMF and introduce the ability for an Ecore package to support variability for the Ecore classes. This concrete adapter achieves this ability by binding elements from the target model (resp. *VariableConceptRole*, *ConceptElementRole*, *elements* and *ConstraintContainerRole*) to elements of the Ecore metamodel (resp. *EPackage*, *EClassifier*, *eClassifiers* and *EPackage*). As a result of these bindings, *EPackageVariableForEClass* and *EPackageVariabilityOperatorForEClass* classes are introduced with their dependent classes and relationships and *EPackage* is extended with a new relationship to *EPackageVariabilityForEClass* (see left part of the figure). Finally, the concrete adapter also contains redefined adaptations (*introduceDescendants1* and *introducedDescendants2*) to select the subset of operators that are appropriate for classes variability.

5 Introducing Variability into SmartAdapters

In [4], we pointed out that aspect reusability is limited in AOM approaches because an aspect model must match exactly the structure of base models and is always woven according to the same rules. To address this issue, we proposed to extend AOM approaches with matching variability and composition variability. This variability was introduced in an ad-hoc way. Supporting these two dimensions of variability in our SmartAdapter approach have been achieved by extending the notion of adapter with the following variability mechanisms:

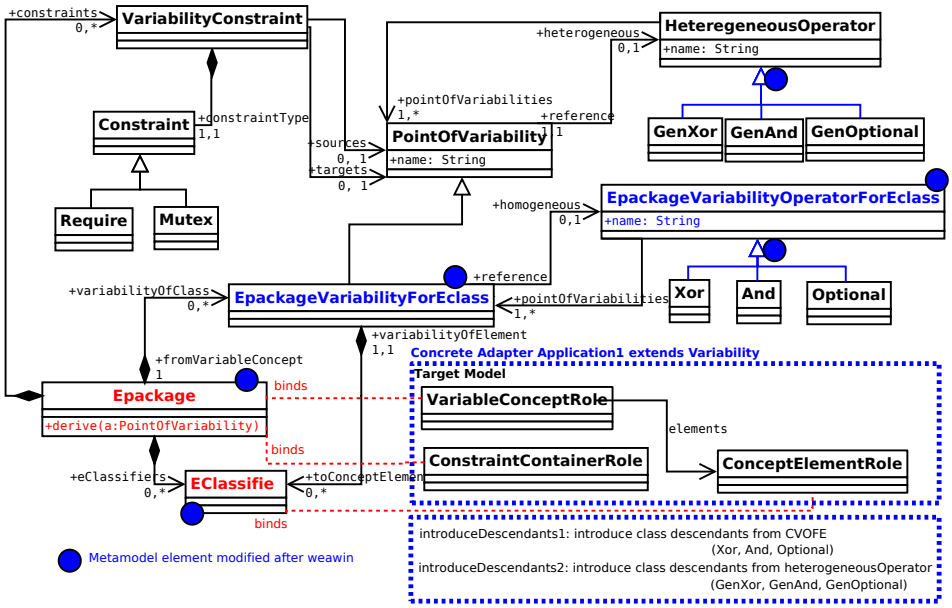

Fig. 6. The variability aspect

- **Optional targets:** in order to specify that some elements from the target model may be present or not in the base model where we want to weave the aspect.
- **Alternative adaptations:** in order to specify that there exists several possible ways to compose the aspect. All the variants are exclusive *i.e.*, we can only choose exactly one variant per alternative.
- **Optional adaptations:** in order to specify that some adaptations of the composition protocol are not mandatory.
- **Constraints between targets and/or adaptations** in order to specify that some variants are dependent or in mutual exclusion. With these constraints, we can ensure the consistency of the composition protocol, after derivation.

Using these mechanisms, a designer can build an aspect model that is adaptable to different contexts. Figure 7 illustrates an aspect model using these mechanisms to integrate the well-know observer pattern into a base model. The target model declares an option to deal with the presence or not of the association between classes playing *SubjectTargetClass* and *ObserverTargetClass*. The composition protocol includes two variants to integrate the classes and association of the pattern into a base model, either by merging or by inheritance.

All the mechanisms presented above can be added to our SmartAdapter approach by applying the previous variability aspect to its metamodel, using SmartAdapter itself. Figure 8 and 9 show the definition of two concrete adapters to achieve this operation. They specialize and complete the abstract adapter of the variability aspect described in Section 4.

The first concrete adapter (*SmartAdapter1*) handles the declaration of options and alternatives within a composition protocol. It binds elements from the

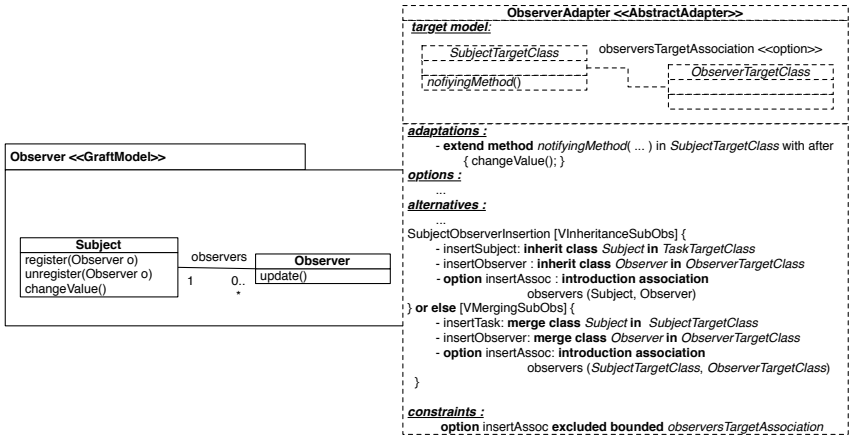

Fig. 7. Example of an aspect model with variability

target model (resp. *ConstraintContainerRole*, *VariableConceptRole*, *ConceptElementRole*, *elements*) to elements of the SmartAdapters metamodel (resp. *Model*, *AbstractAdapter*, *AbstractAdaptation*, *compositionProtocol*). This adapter causes the contextualization of adaptations from the variability aspect with the following results :

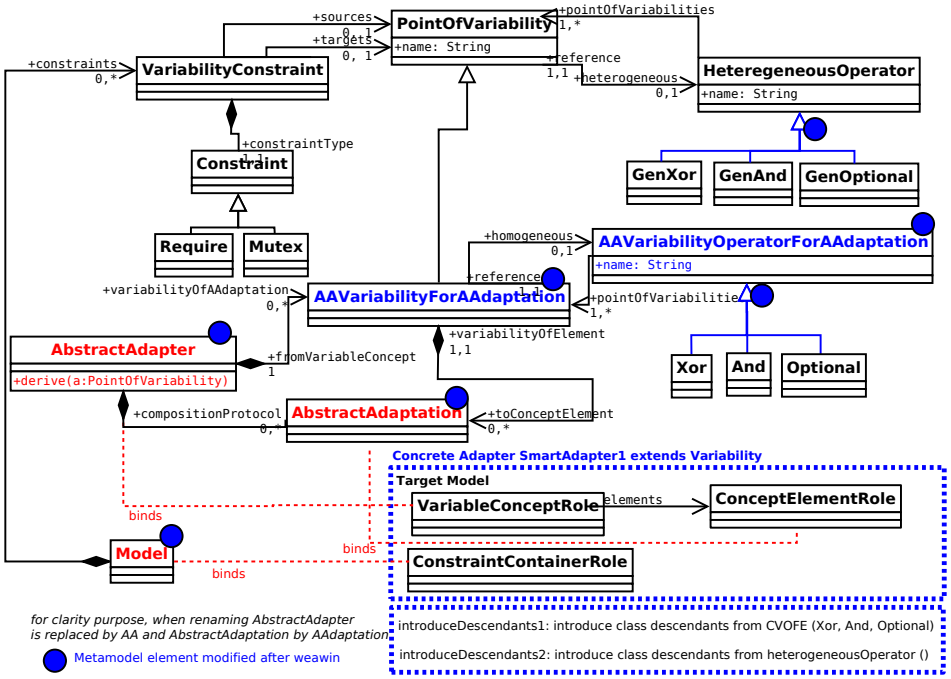

Fig. 8. Adapters for adding variability in SmartAdapters (1)

– the *VariabilityOfElement* class is inserted as a subclass of *PointOfVariability* and this new class is renamed *AAVariabilityForAAdaptation*. This class introduces variability capacities for the *AbstractAdapter* class.
– the content of *VariableConcept* is merged into *AbstractAdapter* class. As a result of this merging, the *AbstractAdapter* is extended with the *derive* method and one aggregation relationship to hold *AAVariabilityForAdaptation* elements.
– the insertion of classes required for describing constraints and operators as well as their relationships.
– the insertion of *AAVariabilityOperatorForAAdaptation* as a superclass for the set of operators (Xor, And, Optional) defined for *AbstractAdapter*.

The second concrete adapter (*SmartAdapter2*) handles the optionality of target elements. It applies the variability aspect to metaclasses of the metamodel representing the target model and its content, by binding *VariableConceptRole* to *TargetModel*, *ConceptElementRole* to *TargetModelElement* and *elements* to *targetElts*. According to these bindings, the *TargetModel* class is extended with the content of the *VariableConceptRole* and with a new relationship to *TMVariabilityForTMElement* that defines the variability for *TargetModelElement*. The operator that can be used for this variability is defined by *TMVariabilityOperatorForTMElement* which is inserted as superclass of *Optional*. Note that classes for describing constraints and operators are only inserted once, even when the aspect is woven in several places.

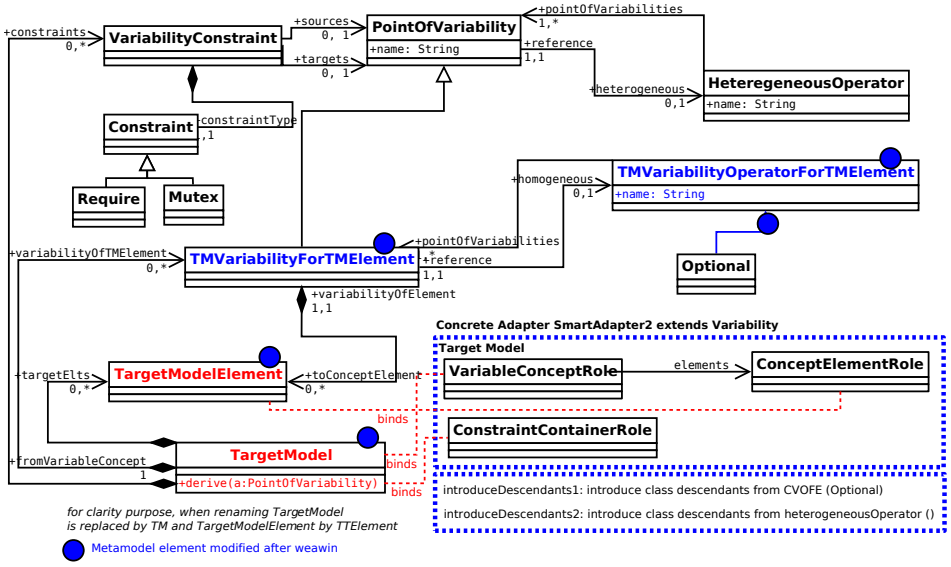

Fig. 9. Adapters for adding variability in SmartAdapters (2)

6 Towards Software Product Line

The previous sections explained how we can flexibly add variability concepts to a domain metamodel, like Ecore or SmartAdapters, in order to be able to easily design models containing variability. In this section, we describe how we derive products (models with no variability, conforming to the former metamodel) from a product line model (model with variability, conforming to the extended metamodel, where the variability aspect has been woven).

As mentioned in Section 2, one of the most practical techniques is *feature modeling* which aims at representing the common and variable *features*[6] (or concepts) of a product family. Feature modeling is not only relevant to requirement engineering but it can also be applied to design or code levels. Hence, every stakeholder can manipulate features "as is", independently of the kind of variability and the level of abstraction. Moreover, feature models (FMs) encourage to define a standard vocabulary for a domain language and are ideal abstractions that customers, experts, and developers can easily understand. FMs hierarchically structure domain concepts into multiple levels of increasing detail thus proposing a taxonomy. When decomposing a feature into sub-features, the sub-features may be optional or mandatory or may form Alternative, Or, or And groups. FMs describe the variability and the commonality of features and represent a set of valid *configurations*. A valid configuration is obtained by selecting features while respecting the parent-child and an intuitive decomposition semantics. Feature models are represented as graph which have a tree-like structure as shown on Figure 10.

In order to take advantage of existing feature-based modeling tools [15], derivation approaches [2, 16] or formal analysis techniques [17, 18, 6], we offer to compute a feature diagram from a model with variability as shown in Section 3. To do so, we use Kermeta [19] which is a metamodelling environment dedicated to Ecore models manipulation. The initial step is to obtain the root feature which corresponds to the root package of the EMF model. Then we traverse the EMF model (which imposes its structure to the feature diagram) by navigating the containment relationships. For each relationship, there are two options:

- **Variability point**: If this relationship contains a variability point (instance of the woven $VariabilityOfElement$ metaclass), we create an operator of the right type (or,xor,vp) according to the operator associated to this variability point. We then retrieve all other instances of $VariabilityOfElement$ which are referenced by the operator in the EMF model. The sub-features are then obtained by forming the union of all $ConceptElement$ instances referenced by the collected $VariabilityOfElement$ instances.
- **"Standard" relationship**: If there is no variability point, we treat the referenced element as a mandatory feature.

Figure 10 shows the feature diagram resulting from the application of the procedure on the EMF model shown Figure 3. Once the feature diagram is built, we can perform product derivation [20, 2, 16]. When a set of features is selected from the feature diagram, the last step consists in actually deriving the product model. For all the selected features, we call the derive operation associated to the

[6] according to [14] a feature is "an increment in product functionality".

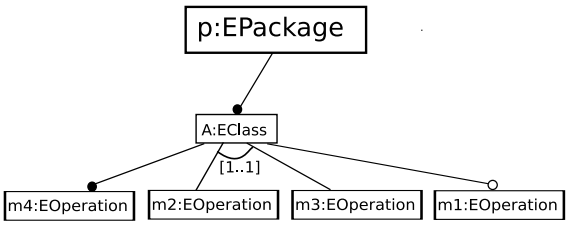

Fig. 10. Computed Feature Diagram for EMF Model with Variability

model element (*e.g.* an instance of `EPackage` in Ecore metamodel or an instance of `AbstractAdapter` in SmartAdapters). This operation is implemented in Kermeta in a generic way, directly in the domain metamodel. It sets the former references with the model elements contained by the point of variability, which is removed. In a second pass, once all the points of variability have been derived, we remove all the remaining points of variability, corresponding to non-selected features. Finally, we can save the product model using the former domain metamodel. This derivation operator is built on top of the Kermeta Model-Development Kit (MDK)[7] for Feature Modeling [2]. We reuse some parts of the Feature Diagram metamodel, some parts of its graphical editor and we have extended the static checker. The derivation process has been designed from scratch.

7 Related work and discussion

We do not address here the composition of models in general but the different approaches to introduce variability into metamodels and models.

Feature modeling is very much adapted to the description of the variability but the hierarchical approach does not provide the expressiveness that is needed and that may be provided by OO modeling approaches. This is the reason why some approaches like [2] use both UML and FMs for modeling a domain. Moreover on the contrary of our approach, the formalism used to describe the domain model contains already the expressiveness for the description of variability.

Other techniques like Ziadi *et al.* extend the UML metamodel in order to include features for modeling variability [20]. Those approaches work at the meta-meta level and extend an existing formalism in order to include variability modeling capabilities. The variability is included in various UML diagrams like class diagram or Sequence diagram. The capabilities introduced in those diagrams are very similar to the ones of our variability aspect which is applied here to class diagram but which could be applied also to other diagrams like sequence diagrams.

The software product line community recently investigated the use of variability techniques to assist the engineering of DSMLs. In [12] the authors propose a metamodel for describing variability which is independant from the models needing variability. In this respect the approach is similar to ours but there are several differences. First they do not aim to compose the two metamodels as we do; on the contrary the metamodel describes only possible substitutions. Second those substitutions are not defined according to the metamodel but to the models and

[7] http://www.kermeta.org

these are the instances which are modified. They promote the idea that variability should not be defined at the metamodel level but at the model level.

In [21] Voelter presents an approach that addresses variability implementation, management and tracing by integrating model-driven and aspect-oriented software development. Features are separated in models and composed by aspect-oriented composition techniques on model level. This approach differs significantly from our approach: the variability is described at the model level with feature models which are transformed in AspectJ source code. They use AOSD as a techniques to compose variants, we use AOM to integrate the variability mechanisms in a domain metamodel.

8 Conclusion and future work

Building families of models related to the same domain is a key issue. It is widely addressed by the SPL community which propose the expressiveness needed by the description of the commonalities and the specificities of each model of the family (i.e. the variability of the family). Variability is a possible orthogonal concern of any domain metamodel and we propose an approach to compose this concern with domain metamodels. The effort of the domain model designer to introduce variability into its models must be reduced as much as possible. We propose to use the SmartAdapters approach which allows i) to minimize the information to be given at composition time and, ii) to guide and control the reuse of the variability aspect in various contexts. A first contribution of this paper is the specification of a variability aspect. We apply this aspect in two different contexts: EMF and SmartAdapters itself. A second contribution is the demonstration that AOM approaches could benefit from the concepts found in SPL without extending their underlying mechanisms, but using only the weaving techniques already present in AOM approaches. We use a version of SmartAdapter without variability (Figure 5) to weave the variability aspect into the SmartAdapter metamodel. We obtain the same expressiveness that the SmartAdapters version with manually introduced variability [4].

This validation of the approach for enhancing domain metamodel (DSMLs) with variability is a first step towards a better modularity in the metamodels. In the short term we aim to reuse this approach in order to introduce other features into DSML (e.g. model checking, editing facilities, etc.) making it more attractive but not more cumbersome especially when these facilities are not needed.

References

1. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
2. Perrouin, G., Klein, J., Guelfi, N., Jézéquel, J.M.: Reconciling automation and flexibility in product derivation. In: 12th International Software Product Line Conference (SPLC 2008), Limerick, Ireland, IEEE Computer Society (2008) 339–348
3. OMG: OMG Unified Modeling Language OMG UML, Superstructure Version 2.2. Technical Report formal/2007-02-03, Object Management Group (2007)

4. Lahire, P., Morin, B., Vanwormhoudt, G., Gaignard, A., Barais, O., Jzquel, J.M.: Introducing variability into aspect-oriented modeling approaches. In: In Proceedings of ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS 07), Nashville, TN, USA (2007)
5. Gray, J., Sztipanovits, J., Schmidt, D.C., Bapty, T., Neema, S., Gokhale, A.: Two-level aspect weaving to support evolution in model-driven synthesis. Addison-Wesley, Boston (2005) 681–709
6. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and A Formal Semantics. In: RE, Minneapolis, Minnesota, USA (2006)
7. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks **51** (2007) 456–479
8. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (1990)
9. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing Cardinality-based Feature Models and their Specialization. Software Process Improvement and Practice **10** (2005) 7–29
10. Griss, M.L., Favaro, J., d' Alessandro, M.: Integrating Feature Modeling with the RSEB. In: ICSR, Washington, DC, USA (1998)
11. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Ann. Softw. Eng. **5** (1998) 143–168
12. Haugen, O., Moller-Pedersen, B., Oldevik, J., Olsen, G., Svendsen, A.: Adding standardized variability to domain specific languages. In: Software Product Line Conference. (2008) 139–148
13. Ramos, R., Barais, O., Jzquel, J.M.: Matching model-snippets. In: In Proceedings of ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS 07), Nashville, TN, USA (2007)
14. Batory, D.S.: Feature models, grammars, and propositional formulas. In Obbink, J.H., Pohl, K., eds.: SPLC. Volume 3714 of LNCS., Springer (2005) 7–20
15. PureSystems. Pure::Variants Website http://www.pure-systems.com/ (2006)
16. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach based on Superimposed Variants. In: Generative programming and component engineering (GPCE). Volume 3676 of LNCS., Springer-Verlag (2005) 422–437
17. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortes, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS). (2007) 129–134
18. Metzger, A., Pohl, K., Heymans, P., Schobbens, P.Y., Saval, G.: Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In: IEEE Conference on Requirements Engineering, Los Alamitos, CA, USA, IEEE Computer Society (2007) 243–253
19. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages. In: Proc. of MODELS/UML'2005. LNCS, Jamaica, Springer (2005)
20. Ziadi, T., Jézéquel, J.M.: Product Line Engineering with the UML: Deriving Products. In: Families Research Book. Springer (2006)
21. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: 11th International Software Product Line Conference, Kyoto, Japan (2007) 10